

Kreiranje korisničkih funkcija za Interbase korišćenjem Borland Delphi-ja

Kada se razvijaju baze podataka za primenu u nekom od informacionih sistema, značajnu ulogu mogu odigrati korisničke funkcije (User Defined Functions, u daljem tekstu UDF). Ovaj domen razvoja i dizajniranja baza podataka, iako retko upotrebljavan od strane projekatara, često može doneti mnoge olakšice prilikom implementacije komplikovanih SQL upita u kojima je potrebno ugrađivanje komplikovanih matematičkih funkcija.

Interbase je jedan od sistema za upravljanje bazama podataka koji podržavaju UDF na jednostavan način, a njihova konkretna implementacija zavisi od platforme operativnog sistema na kojoj se server izvršava. Pošto je tema izrada UDF-a u alatu Delphi, smatra se da se server izvršava na platformi MS Windows gde se UDF smešta u dinamički linkovanoj biblioteci (dll).

Da bismo bili u mogućnosti da koristimo UDF, moramo nakon izrade odgovarajuće biblioteke izvršiti prijavljivanje biblioteke u okviru baze. Ovo ćemo uraditi tako što biblioteku iskopiramo u poddirektorijum <Udf> koji se nalazi u glavnom direktorijumu gde je smešten server, a zatim iz konzolne aplikacije pokrenemo sledeći skript:

```
DECLARE EXTERNAL FUNCTION name [Paramter list]
RETURNS {<datatype> [BY VALUE] | CSTRING (int)}
ENTRY_POINT "<entryname>"
MODULE_NAME "<modulename>"
```

Prođimo postupak izrade i prijavljivanje funkcije kroz mali primer. Posmatrajmo, na primer, slučaj da vam je potrebno da često u okviru nekih izveštaja koje izrađujete u klijentskoj aplikaciji prikazete dan kada su izveštaji rađeni. Da biste sebi olakšali posao, najlakše je to odraditi na sledeći način:

1. Otvorite u Delphi-u nov projekat za izadu dinamičkih biblioteka (File | New | DLL).
2. Napravite sledeći kod i snimite projekat:

```
library udfs;
uses SysUtils, Classes;
function UDF_DayOfToday : PChar; cdecl;
begin
Result := PChar(LongDayNames[DayOfWeek(now)]); { Vraćanje trenutnog dana
u nedelji }
end;
exports UDF_DayOfToday;
```



```
begin  
end.
```

3. Prevedite biblioteku (Ctrl+F9), presnimite je u predhodno navedeni direktorijum, a zatim pređite u konzolu baze podataka i u interactive SQL prozoru ukucajte sledeći skript:

```
DECLARE EXTERNAL FUNCTION UDF_DayOfToday  
RETURNS CSTRING(12)  
ENTRY_POINT "UDF_DayOfToday" MODULE_NAME "udfs"
```

Primenite zatim naredbu "commit" da bi se uneseni skript i fizički smestio u bazu podataka. Ovim je funkcija spremna za upotrebu i možete je odmah testirati. Dobar primer za testiranje novih stvari je korišćenje pokaznih baza podataka kao što je baza "employee" koja se isporučuje zajedno sa Interbase serverom. Ukoliko ste registrovali UDF za pomenutu bazu ukucajte i izvršite sledeći skript:

```
select first_name, last_name, UDF_DayOfToday() from employee
```

Kao rezultat trebalo bi da dobijete nešto slično sledećem prikazu, ukoliko su vam ista podešavanja u okviru Control Panel-a za prikaz dana u nedelji:

FIRST_NAME	LAST_NAME	UDF_DAYOFTODAY
Robert	Nelson	ponedeljak
Bruce	Young	ponedeljak
Kim	Lambert	ponedeljak
Leslie	Johnson	ponedeljak
...

Kao i većina sistema za upravljanja bazama podataka i Interbase je pisan u programskom jeziku C/C++ tako da se kao osnovi tip promenljive za rad sa znakovnim nizovima ne koristi String kao u Pascal-u, nego tip PChar što predstavlja znakovni niz koji se završava sa ASCII kodom 0. Kada se iz Interbase-a želi prosleđivati kao parametar u funkcijama, mora se navesti kao tip CString(len), gde je len dužina prosleđenog parametra. Sledeći primer prikazuje tehniku rada sa takvim tipovima podataka iz programskog okruženja Delphi:

```
function UDF_Left(var pStr: Pchar; var Len: integer; var pStrResult: PChar):  
PChar; cdecl;  
var  
i: integer;  
begin  
result := szRes; { Pošto se radi o pokazivačkom nizu proglašićemo da result  
funkcije ukazuje na istu memoriju kao i promenljiva pStrResult }
```

```

i := 0; { Inicijalizacija brojača }
while (pStr^ <> #0) and (i < Len) do begin { Postavljanje uslova za kopiranje
karakera iz niza }
pStrResult^ := pStr^; { Kopiranje karaktera }
Inc(i); { Inkrementiranje brojača, i znakovnih nizova }
Inc(pStr);
Inc(pStrResult);
end;
pStrResult^ := #0; { Dodavanje nultog ASCII koda na kraju niza }
end;

```

Ovde je prikazana tehnika rada sa znakovnim nizovima koji su terminisani nultim znakom. Ukoliko obratite pažnju na zaglavlja, funkcija u oba slučaja je korišćena direktiva `cdecl` koja označava da će se prosleđivanje parametara između funkcija izvršiti po deklaraciji kakva je data u jeziku C. Ukoliko želite više informacija o tome, možete se informisati u Delphi-evom sistemu pomoći. U ovim primerima je prikazano kako je moguće obrađivati znakovne nizove koji se prenose kao parametri u korisničkim funkcijama. Ovakav način prosleđivanja parametara nije u duhu jezika kao što je Object Pascal, ali se vidi da je na lak način moguć rad i sa takvim parametrima, što omogućava rešavanje velikih problema na serveru, a ne prevlačenje nepotrebnih podataka sa servera baze podataka ka klijentima i onda vršenje obrade, ili pisanjem komplikovanih i zamršenih SQL rečenica. Da biste isprobali ovu funkciju nakon prevođenja, pod uslovom da ste ovu funkciju dodali u prehodnu biblioteku, ispišite sledeći skript za prijavljivanje funkcije u bazu:

```

DECLARE EXTERNAL FUNCTION UDF_Left
CSTRING(64), INTEGER, CSTRING(64)
RETURNS PARAMETER 3
ENTRY_POINT "UDF_Left" MODULE_NAME "udfs"

```

Sledeće na šta je potrebno obratiti pažnju prilikom pisanja UDF funkcija je više nitna organizacija rada Interbase-a. Ne preporučuje se korišćenje globalnih promenljivih u bibliotekama tako da ovako, na primer, dobijete grešku "...attempted to access a virtual address without permission", znajte da vam Windows operativni sistem vraća poruku kako je je vaš višenitni program pokušao da pristupi resursima koji nisu bezbedni za pristup iz dotičnih programa. Ukoliko vi ipak želite da kreirate privremene promenljive koje ćete kasnije upotrebiti, a sigurni ste da neće doći do nepoželjnih posledica, obratite pažnju da u delu za inicijalizaciju u vašoj UDF postavite globanu sistemsku promenljivu `IsMultiThreaded` na vrednost `true`. Dobro rešenje je da funkcija vraća dinamički kreirane rezultate, a ne preko parametara kako je prikazano u prethodnom primeru. Prethodni primer je urađen na dotični način samo radi pokazivanja jedne od tehnika. Kada uzmemo da analiziramo šta se događa sa memorijom na serveru, videćemo da se, nakon velikog broja poziva funkcija, slobodna memorija na serveru smanjila. Ovde se radi o sledećoj problematici.

Kada je naša funkcija vratila rezultat, privremeno je kreirana instanca rezultata i predata vrednost pokazivača na datu memorijsku lokaciju. Pošto je prilikom pisanja funkcije data direktiva cdecl koja označava da kao rezultat nakon preuzimanja iz memorije, treba da oslobodi program koji je izvršio poziv date funkcije, naša generisana biblioteka to ne radi. Od verzije Interbase-a 5.0 implementirana je direktiva free_it koja služi da bi se eksplicitno reklo Interbase-u da počisti memoriju nakon preuzimanja rezultata. Tako da će sada naša deklaracija UDF-a izgledati na sledeći način:

```
DECLARE EXTERNAL FUNCTION UDF_Left  
CSTRING(64), INTEGER  
RETURNS CSTRING(64) free_it  
ENTRY_POINT "UDF_Left" MODULE_NAME "udfs"
```

Ovde postoji podela u mišljenjima ko i kada treba da vrši čišćenje memorije koja je upotrebljena za vraćanje rezultata. Jedna grupa tvrdi da dodate funkcije u sistem ne bi trebalo da vrše “prljanje” memorije, ili ako to urade da onda iza sebe i “počiste” svoje smeće. Moje lično mišljenje je da je logičnije da “čišćenje” memorije radi program koji poziva funkciju, jer to radi isključivo nakon preuzimanja rezultata. Ovde smo pomalo zagazili u velike probleme dodeljivanja memorije i kreiranja višenitno bezbednih funkcija. U Borland-u su, nakon izlaska verzije 2.0 Delphi-a, primetili kako se upravljanje memorije u Windows operativnom sistemu ne obavlja na najbolji način, tako da je uvedena promenljiva IsMultiThreaded kojom programer eksplicitno govori sistemu da je softver koji se pravi višenitno orijentisan. Ovo ne mora da bude obavezno, ali je poželjno u svim slučajevima izrade ovakvih softvera kod kojih se ne koristi klasa TThread i važi za sve unit-e u projektu.

Kako Interbase prilikom poziva UDF-a ima već kreirane niti, nije prekinuto pravilo da je krajnje nepoželjno praviti višenitne biblioteke. Da bi ovakve biblioteke napravile sekcije koda kao što su initialization i finalization koje mogu rešiti date probleme, međutim da bi se ovakvi problemi rešili na prilično efikasan način potrebno je [X1]veće zadubljanje u upravljanje memorijom u bibliotekama koje trebaju biti više nitno bezbedne. Setimo se čemu nam trebaju koristiti ovakve biblioteke. Glavni cilj nije napraviti čudovište koje radi svašta sa memorijom na serveru baze podataka, već sadrži male i jednostavne funkcije u kojima nam je bilo znatno lakše da implementiramo određene funkcije, nego da pišemo komplikovane SQL rečenice. Shodno tome treba voditi računa da prilikom pisanja ovih biblioteka ne učinimo sebi medvedu uslugu ili da ne zahtevamo previše od sistema za upravljanjem bazama podataka. Pisanjem komplikovanih UDF-a dovodi se u pitanje validnost projekta datog informacionog sistema i zahteva od servera baze podataka i lako je moguće da dođe do određenog “curenja” u memoriji ili do pada servera baze podataka što nikako nije cilj, tako da o se ovome mora voditi računa, a i pisanje ovakvih biblioteka pomalo prevazilazi nezavisnost koda od promene operativnog sistema što u svakom slučaju ne bi trebalo da bude, već je poželjno da se prave takve biblioteke koje je moguće na lak način premestiti na drugi operativni sistem.



Ovde je prikazana tehnika pravljenja UDF-a i skrenuti je pažnja na probleme koji se mogu javiti prilikom pravljenja ovakvih biblioteka. Načelno ne postoji loša strana koja bi naterala projektanta sistema da izbegne izradu ovih biblioteka, pod uslovom da se vodi računa o jednostavnosti funkcija i njihovoj atomičnosti. Prikazani primeri su izrađeni u Delphi-u, ali je identična izrada ovih biblioteka u Kylix-u, sa tom razlikom da se biblioteke izrađuju kao Shared Object (SO) što je pod Unix operativnim sistemima analogija za dll-ove, tako da je ovime pokrivena i platforma Linux operativnih sistema, što bi trenutno bila optimalno rešenje što se tiče odnosa (performanse / cena) sistema.